Section 28
Ground Time Prediction Subsystem

The *Ground Time Prediction Subsystem (GTPS)* provides improved predictions of aircraft ground times to the *Flight DataBase Processor (FDBP)*. It provides a ground time prediction for each flight that the *FDBP* requests. The *GTPS* bases its predictions upon the historical flight data that it collects and processes. As a secondary feature, the *GTPS* provides to requestors, through its *Delay Advisor* (see Section 21) sub-function, reports of the historical performance of flight ground times.

Design Issue: Batching of Data Processing

Insofar as it is practical, data processing has been concentrated in batch mode processes so that the real-time continuous processes have less work to do.

Design Issue: Two modes of generating ground time predictions

The *GTPS* has two significant modes of generating ground time predictions. (It also allows defaults in special cases.) One mode generates predictions based on a flight identifier (FID). The other mode generates predictions based on a category (Cat). The *GTPS* is designed to prepare both modes whenever possible so that when the *FDBP* requests a prediction, the *provide_gtp* subfunction selects the most appropriate prediction.

Design Issue: Coordinated Universal Time (UTC)

Except where noted, all times used in the *GTPS* are UTCs, expressed in minutes after UTC midnight. All dates are expressed in *Julian days*, a representation in which the date is expressed as elapsed days from the commencement of January 1, 1980. January 1, 1980, is considered day 0, and subsequent days are enumerated consecutively to the current day.

Exceptions are that UTC hour and minute time formats are used in the **cat_def_file** and in the **delay_advisor** reports. Month and day date formats are used in the **delay_advisor** reports.

Processing Overview

The *GTPS* (see Figure 28-1) supplies the *FDBP* with a predicted ground time and predicted time enroute for a specific flight, upon request. The request will be made some time prior to the flight's departure. After flight completion, the predicted ground time value is:

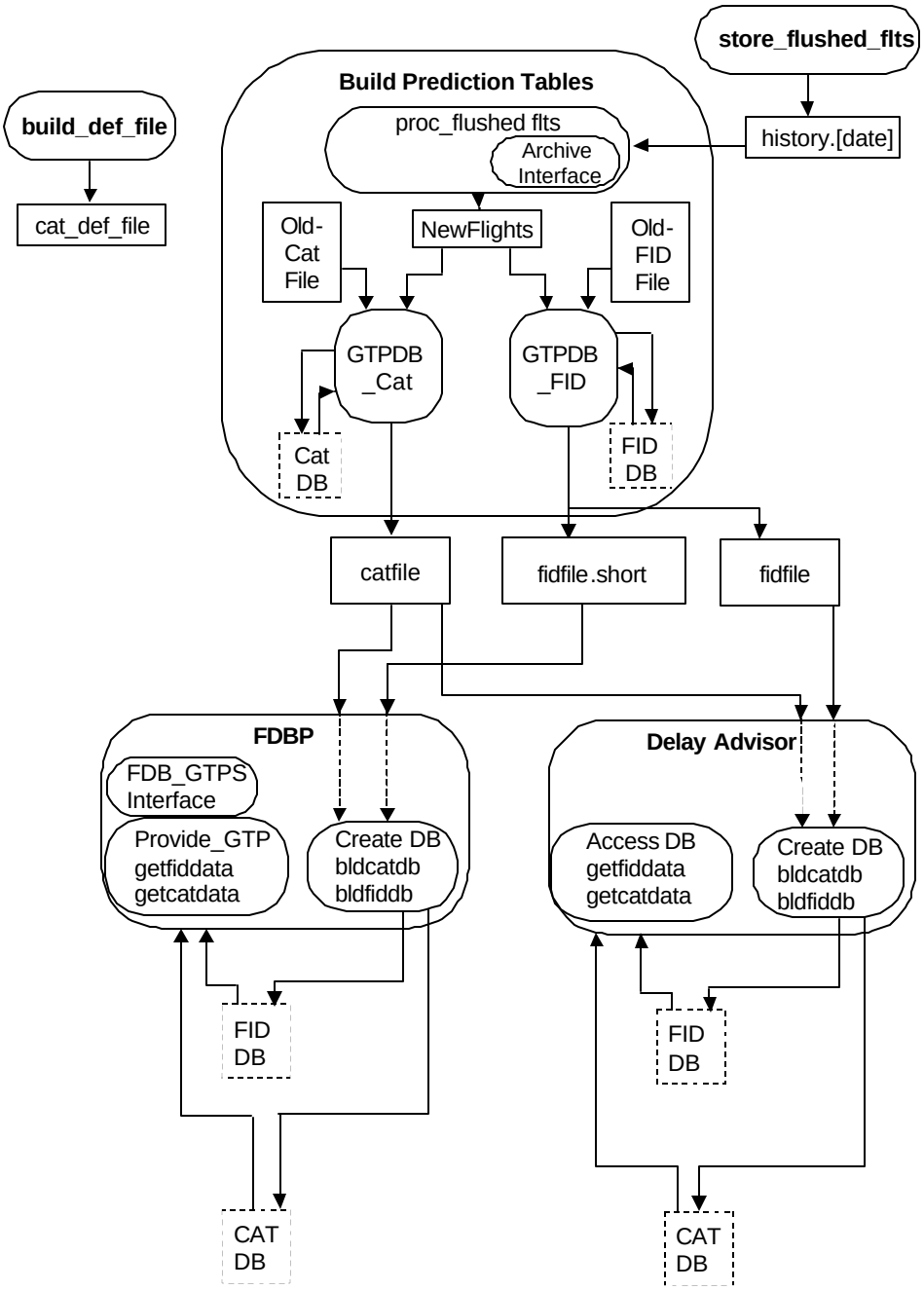

Figure 28-1. Data Flow of the Ground Time Prediction Subsystem

- Flushed by the *FDBP*
- Collected by the *GTPS*
- Processed by the *GTPS*, along with other flight record information so that it helps form subsequent predictions
- Made available for users to view via the *GTPS* Delay Advisor

This process can be represented as several ETMS subfunctions.

The *GTPS* uses the following subfunctions or processes, each of which is executed by one or more processes or routines:

- The *store_flushed_flts* subfunction receives flushed flight records from the *FDBP* and stores a subset of each flushed record in an archive of history files.
- The *build_prediction_tables* process takes history files and uses them to create an FID-based and category-based database of ground time information for use by the subsequent subfunctions.
- The *provide_gtp* subfunction is incorporated into the *FDBP*. It provides ground time predictions for flights as the *FDBP* needs them. It takes those predictions from a category-based or an FID-based database in accordance with an algorithm defined later in this section.
- The *build_def_file* ancilliary process generates a static table of information for each of the key airports for which the category database provides information. This table of information is stored in the **cat_def_file,** which is needed by all the subfunctions except *store_flushed_flts.*
- The *delay_advisor* subfunction provides to *TSD* users ground time information reports about both specific individual flight departures and about the ensemble of flights previously departed from a specific airport. See Section 21 for a detailed description of the *Delay Advisor.*

The *store_flushed_flts, delay_advisor* and *provide_gtp* subfunctions operate continuously. The *build_prediction_tables* subfunction operates in batch mode (nominally once weekly). The *build_def_file* ancillary subfunction is executed only as needed, such as to replace a lost **cat_def_file** or to generate a changed one.

28.1 The Store_flushed_flts (Sff) Subfunction

The *Store_flushed_flts (Sff)* subfunction collects flight records of departed flights as they time out and are flushed from the *FDBP* and stores them in a history file archive composed of ASCII files. These ASCII files are then available for use as inputs to the programs that build the *GTPS* flight and category databases.

Processing Overview

The *Sff* subfunction consists of three processes: *Sff receiver, Sff collect,* and *Sff manager*. The *Sff receiver* process receives flushed flights from the *FDBP*. The *Sff collect* retrieves the flushed flights and stores them in ASCII files. The *Sff manager* acts as a framework for controlling the other two processes. Figure 28-2 illustrates the process flow.

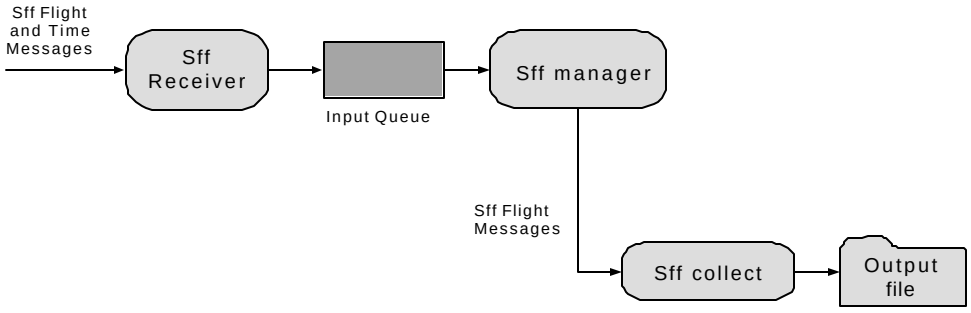

Figure 28-2. Data Flow of the Store_flushed_flts Subfunction

28.1.1 The Sff Receiver Process

Purpose

The *Sff receiver* process queues the flight and time data from the *FDBP*.

Execution Control

Upon initialization, the *Sff manager* process starts the *Sff receiver* as a child process. This data-driven process runs continuously; if the *Sff manager* process fails, the *Sff manager* parent process restarts it using the old queue so that a minimum amount of data is lost. Non-

fatal errors cause an error message to appear in the process window of the ETMS operator node.

Input

The *Sff receiver* input consists of flight data (without the event list) and time data from the *FDBP*.

Output

The Sff *receiver* output is identical to its input.

Processing

The *Sff receiver* creates a socket in order to receive data from the *FDBP*. The *Sff receiver* then allocates memory for use as a queue. As messages arrive via the socket, the *Sff receiver* puts them in the queue for use by the *Sff collect* process when it is ready. See Section 23.2 for a more detailed description of receiver processes.

Error Conditions and Handling

In general, if the files necessary for creating the queues are locked, *Sff receiver* unlocks them before creating or opening them.

28.1.2 The Sff Collect Process

Purpose

The *Sff collect* process stores as ASCII files a subset of the *FDB* flight data received by the *Sff receiver.*

Execution Control

Upon initialization, the *Sff manager* starts the *Sff collect* process. This data-driven process runs continuously. Non-fatal errors cause an error message to appear in the process window of the ETMS operator's node.

Input

The *Sff collect* process input consists of flight data (without the event list) from the *FDBP*. Refer to Table 28-2 for a detailed description of this input.

Output

Sff collect output is a subset of its input. Refer to Table 28-2, the *history_file* data structure, for a detailed description of this output.

Processing

Sff collect retrieves flight data from the *Sff manager*. It increments the count of the number of flight data processed and then writes a subset of the flight data to an open file called *history.[date-time]*.

Error Conditions and Handling

Non-fatal errors are written to the process window of the ETMS operator's node.

28.1.3 The Sff Manager Process

Purpose

The *Sff manager* process directs the functional flow of processing from initialization through storage of flight data. It also opens and closes output files based on time data received from the *FDB* and writes general statistics and top level errors to the process window of the ETMS operator node.

Execution Control

The *Sff manager* starts the *Sff receiver* and *Sff collect* processes. The *Sff manager* monitors the time received from *FDB* and opens and closes the archiving files according to the user's parameter. It also monitors the size of data in the process window, closes the window, and opens a new one when necessary. The *Sff* is started by the Hubsite operations staff.

Input

The *Sff manager* input consists of an input parameter file called *store_flushed_flts.params* containing the following four parameters:

(1) working directory - the pathname of the directory in which *Sff* will run

(2) archive directory - the pathname of the directory in which to open the files used for storing the flight data

(3) *Sff receiver* parameter file - the name of the file containing the parameters for establishing the receiver

(4) file time length - an integer that represents the number of whole hours of data to be stored in each output file

Output

Sff manager output is any error or informational message and the statistics it writes to the process window of the ETMS operator node.

Processing

The *Sff manager* begins by reading the input parameters (see **input** above for description). The *Sff manager* then either generates *Sff receiver* if it does not exist or links to one if it does exist. The *Sff manager* then opens an output history file (called *history.[date]*) for use by *Sff collect*.

The *Sff manager* retrieves data from the receiver and determines the data type. If the data is flight data, the *Sff manager* passes the data to *Sff collect;* if the data is time data, the *Sff manager* checks the *FDB* time against a previously determined file and closes the file at a specified time. It immediately opens a new file for use by *Sff collect*, increments the appropriate counts, and writes statistical data on a regular basis to the process window of the ETMS operator node.

Error Conditions and Handling

Errors are written to the process window.

28.2 The build_prediction_tables Subfunction

The *build_prediction_tables* subfunction produces the category and FID database files that are used to support the *delay_advisor* and *provide_gtp* subfunctions. These database files each hold predictions of the ground time for the next instance of a specific flight or a category of flight. They also hold data used to build those predictions. The *delay_advisor* subfunction reports supporting data to users upon request. The *Provide_gtp* subfunction selects an appropriate prediction and gives it to the *FDBP*. The *build_prediction_tables* processes are run entirely in batch mode.

Input

The inputs are:

- A history file of flushed flight records. A history file from *store_flushed_flts* and the previous versions of the **catfile** and **fidfile** provide data inputs that the *build_prediction_table* subfunction converts to a new **catfile**, **fidfile.short**, and **fidfile**.
- Old or previously generated **fidfile** and **catfile**
- The **cat_def_file**, an input that is used to help *build_prediction_tables* in categorizing flights

The history files and the **cat_def_file** are ASCII files. The previous versions of **catfile** and **fidfile** are binary files.

Output

The **catfile** and **fidfile** are the outputs, which are binary files. An additional output is **fidfile.short** which has the same data structure as **fidfile**.

Processing

The *build_prediction_tables* process takes flushed flight data available in the history files and converts it into the format of a FID-based and a category-based database. As part of the processing, the previous category and FID databases are recycled and combined with new data recently collected in the history files. The combination involves algorithms described in the processing sections of *gtpdb_fid* and *gtpdb_cat* described later in this section.

Processes Comprising build_prediction_tables

The *build_prediction_tables* subfunction is implemented by the following three processes, which are described separately on the following pages:

(1) *Process proc_flushed_flts*

(2) *Process gtpdb_fid*

(3) *Process gtpdb_cat*

(4)

28.2.1 The proc_flushed_flts Process

Purpose

The *proc_flushed_flts* subfunction performs the following tasks:

(1) Takes a history file that consists of a record of data on each flight

(2) Discards records that are egregiously incomplete

(3) Winnows extraneous data

(4) Computes some new values derived from the data

(5) Assigns flights to categories

(6) Puts the resultant processed data into the data structure **newflights** file for use by subsequent *GTPS* processes

Execution Control

The *proc_flushed_flts* process is run in batch mode, at a nominal frequency of once a week by entering the following command string:

<executable name (e.g., proc_flushed_flts.exe)><parameters file (e.g., param_pff)> <history file name> <user specified output file name>

Input

The two inputs are the **cat_def_file** and a history file. At its initialization, the process reads into its memory data from the **cat_def_file,** pointed to by the parameters file**,** and then it reads flight records one at a time from a user-specified history file.

Output

The only output is the data structure **newflights** file in binary format.

Processing

The *proc_flushed_flts* subfunction loads data from the **cat_def_file** as it starts up. It then performs the following tasks:

(1) Opens the user-designated history file and points to the first record

(2) Reads flight record data, one flight record at a time

(3) Discards records that are incomplete or that fail consistency tests or tests of reasonable values

(4) Performs processing applicable to a record-at-a-time in its *single_record_processing* module

(5) Writes the record to a newly opened **newflights** file once all processing is completed on a record and the record survives the discard operation

(6) Counts the number of records in **newflights**

(7) Appends that number as the first record of the file for future processing convenience

(8) Closes the **newflights** file, which stops the process

Figures 28-3, 28-4, and 28-5 illustrate this processing. They show an overview of *proc_flushed_flts*, the detail of the key routines within *proc_flushed_flts* (which are *discard_inadequate_records* and *single_record_processing*). Figure 28-6 shows the detail of a key routine within *single_record_processing* (which is *compute_cat_table_specific_vars*).

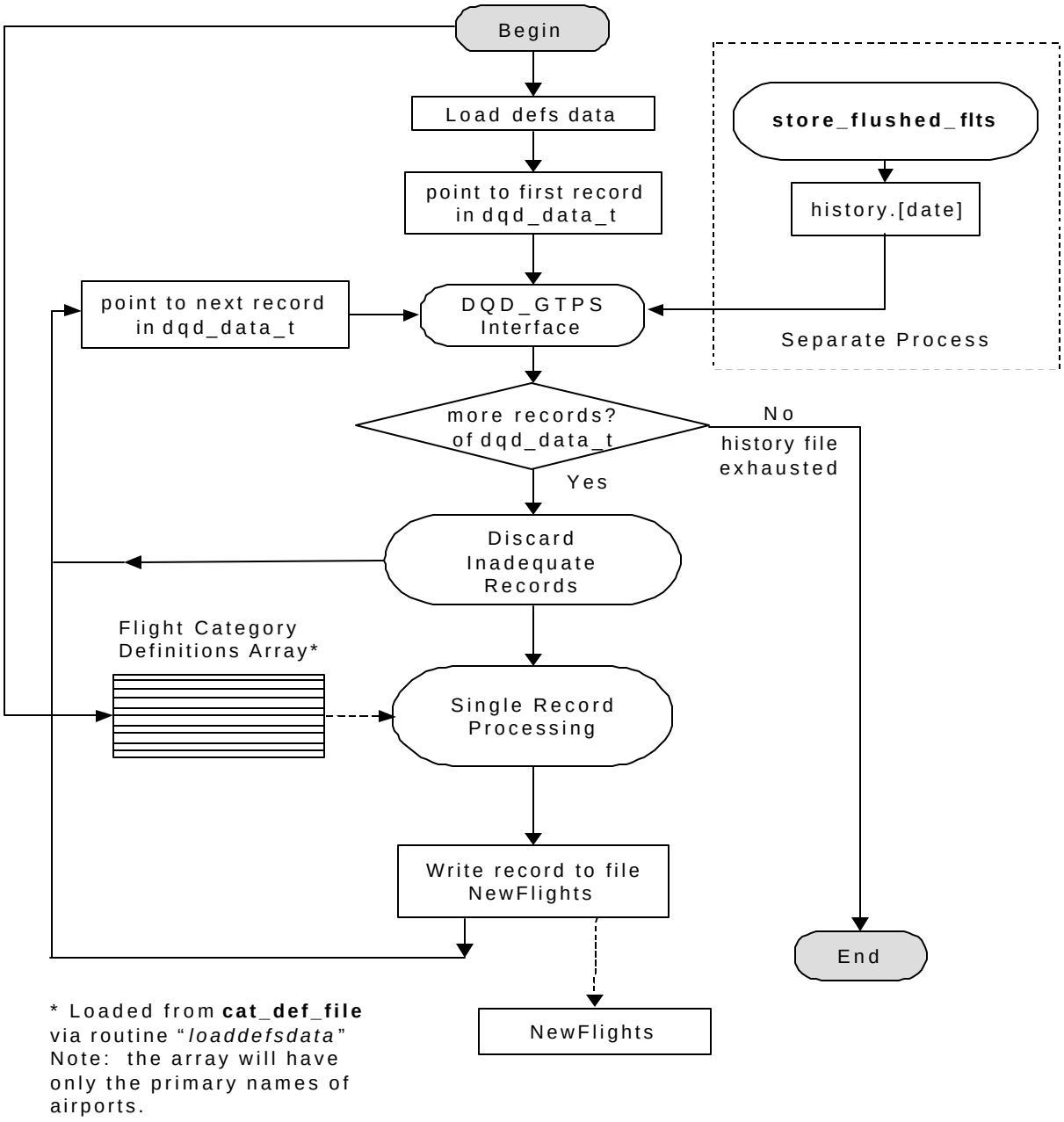

Figure 28-3. Logic for the proc_flushed_flts Process

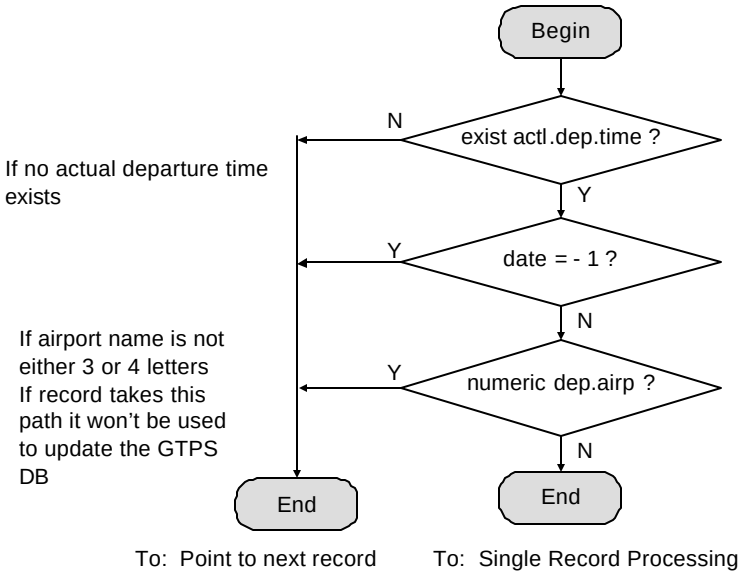

Figure 28.4. Logic for the discard_inadequate_records

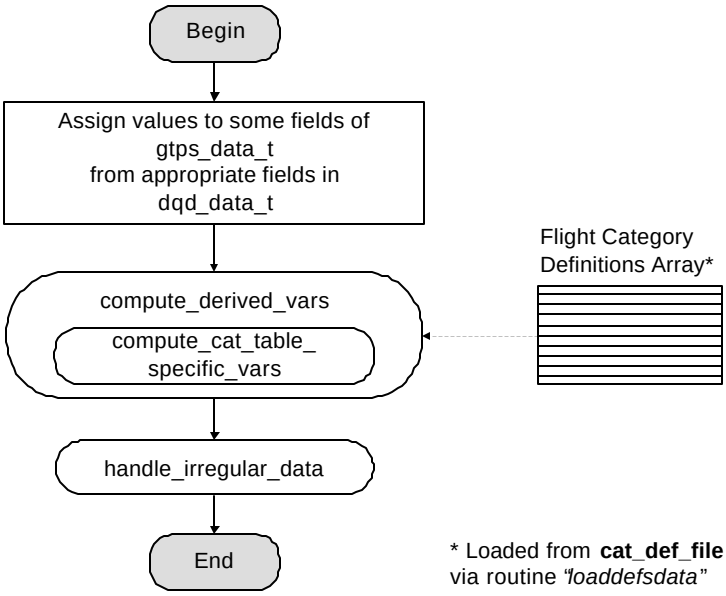

Figure 28-5. Logic for single_record_processing

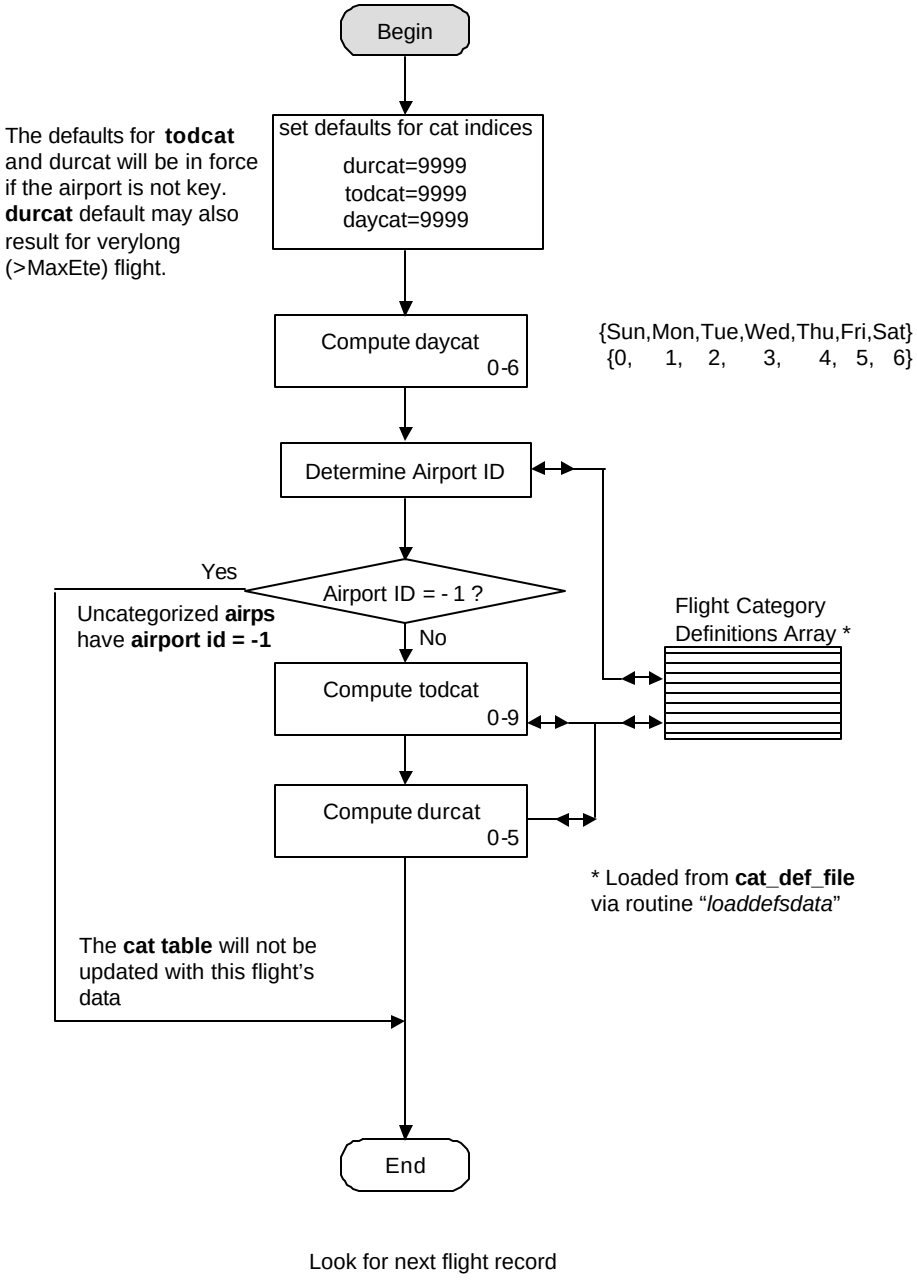

Figure 28.6. Logic for compute_cat_table_specific_vars

Error Conditions and Handling

Errors are reported to the user's screen along with a diagnostic message. If the error is terminal, the program exits, allowing the user to investigate and correct the problem.

28.2.2 The gtpdb_fid Process

Purpose

The *gtpdb_fid* process creates or revises a database of flights, which is written as a **fidfile**. For each flight, the process makes a prediction of the ground time expected of the next instance of that flight to depart. These predictions are included in the **fidfile**. The process also writes a more succinct version of the database, named **fidfile.short**.

Design Issues

The flight identifier, or FID, database is maintained as an array of linked lists, with each array element holding information pertaining to one flight. The linked list associated with the array element holds instance-specific information for each instance of this flight that has occurred. As many as 14 instances are retained. If more than 14 are received for a particular flight, the oldest instances are discarded first, to make room for the new instances.

After the FID database has been prepared and revised as needed by *gtpdb_fid*, the process writes the entire array of linked lists to a permanent medium file (on disk) and names it **fidfile**. The **fidfile** is needed for restoring the FID database to memory. It is also needed to support the *delay_advisor* application. However, it is possible to use a much shorter file to support the FDBP application. It is sufficient for the FDBP to have only the most recent instance data, and not the previous 13 held in the FID database. Therefore a **fidfile.short** is written in parallel with **fidfile** by *gtpdb_fid*. **Fidfile.short** contains only the most recent instance per flight.

Execution Level

This process is run in batch mode, at a nominal frequency of once weekly. It is run by entering the following command string:

<executable name (e.g., bpt_fid.exe)> <parameter file (e.g., param_algor)> <restoration file name (e.g., old_fidfile)> <revision file name (e.g., newflights)>
<path name for directory for output files>

If restoration is not needed, the keyword "none" should be used in place of the restoration file name, such as in the following example:

<executable name (e.g., bpt_fid.exe)> <parameter file (e.g., param_algor)> none <revision file name (e.g., newflights)> <path name for directory for output files>

Input

The process takes the following input files:

- A parameters file, with values for the algorithm parameters, in Table 28-1
- A previous output file from this process, which is an old version of the **fidfile**. This input file is used to restore the FID database as it was at the end of the last execution of the process.
- A **newflights** file, which is used to revise the restored database in accordance with data newly collected since the last execution of the process

Output

The process outputs are the **fidfile**, which is an updated binary file of the last 14 instances of flight-by-flight data, and **fidfile.short**, which is like **fidfile** except it only holds the most recent instance per flight.

Processing

This process maintains and updates a history of ground time information of the most recent 14 instances of every flight departing each NAS airport. For each flight, the process generates a prediction, **Hegt**, for the ground time expected of the next instance of the flight. (For each flight, gtpdb_db also attempts to generate a prediction, **Hete**, of flight enroute time based on the history of enroute times of prior instances of the flight. This enroute time is not currently used.)

The process divides its work into the following three tasks:

- Restores the FID database from the **fidfile**, which had been previously generated
- Revises the FID database with new flight data recently collected in **newflights** file

- Writes the revised database to a new **fidfile** and to a **fidfile.short** Stale flights are not written, and hence disappear when the process terminates. Stale flights are those for which no instance has been observed in **MaxFidAge** days (presently **60**).

An overview of the *gtpdb_fid* process is presented in Figure 28-7, following the text description of the algorithms. Figures 28-8 and 28-9 show more detail about the *revise_fid_db* sections of the process.

Prediction of Ground Time Based on FID

The core of the process work occurs in the Revise section of code, in which flight-specific, FID-ased, ground time predictions, denoted **Hegt**, are made for each specific flight for which some history has been obtained. FID-based predictions are made for flights from all airports.

The algorithm requires the use of the previous value of **Hegt**. A true previous value does not exist for the first occurrence, therefore, we artificially assign to the previous value the constant value **InitHegt**. The current value of **InitHegt** is **15**.

At startup time, the algorithm reads a file **param_algor** that contains parameters to compute the new predictions of the ground time. Table 28-1 lists the currently recommended parameter values for this file.

Table 28-1. Algorithm Parameters

(all units in minutes)

Upper Threshold for Adjustment	**T1**	**3**
Lower Threshold for Adjustment	**T2**	**-2**
AGT Out of Range	**T3**	**45**
AGT – EGT Out of Range	**T4**	**15**
Step Size for Adjustment	**T5**	**-1** **+1**

As flights are flushed and their information is available to update the *GTPS* FID database's **Hegt** predictions, each new flight's actual ground time (**Agt**) is integrated with the previous value of **Hegt**. Even the information from the very first occurrence of a flight is used to

update the **Hegt** prediction, using **InitHegt** as the previous value as noted above. The update algorithm follows.

Update Algorithm: The difference is computed between **Hegt**, the current predicted ground time for pending flights, and **Agt**, the actual ground time experienced by the recently flushed instance of the flight.

Difference = Agt - Hegt (units of minutes)

The **Difference** is used to set a value for an **Addend** as shown in the following table:

If **Difference** >= T1, then **Addend**= T5

If **Difference** <= T2, then **Addend**= -T5

The **Addend** is then used to adjust the existing **Hegt** to form the new **Hegt**, as follows:

Hegt [pending] = **Hegt** [previous] + **Addend** (units of minutes)

The following classes of flights are not used to update the FID database predictions of ground times:

- Controlled flights
- Flights with a negative actual ground time, **Agt**
- Outliers, flights with **Agt** exceeding **T3**

If the first occurrence of a flight has an **Agt** above **T3**, currently assigned the value 45 minutes, the "previous value" is given as usual by **InitHegt**, but that Initial value is not updated by the outlying **Agt**. Subsequently occurring flights which have **Agts** above **T3** are not used to update the **Hegt**. The **Hegt** associated with their flight is not updated.

- Flight instance with the actual ground time exceeding the predicted ground time, **Hegt**, by greater than **T4** minutes, **Agt > Hegt +T4.** This instance is going to be check only if the **FidCount** is greater than **MinNumFid.**

However, when **Agt > Hegt + T4** is true, the algorithm checks the previous flight with the same expression. If the test of the previous flight returns true, it means that is an outbreak. Then this flight instance is used to update the ground time prediction **Hegt** with the addend.

T4 functions like **T3,** except it is the upper limit not on **Agt** but on **Agt-Hegt.**

It should be noted that these flights are not excluded from the database. The database retains them for presentation to the user via the Delay Advisor. They simply are not allowed to update the **Hegt** prediction.

Prediction of Enroute Time Based on FID

Ete is used to denote the actual time enroute (as measured by the difference between times in **DZ** and **AZ** messages) of a specific flight instance. **Hete** is used to denote the prediction made by *gtpdb_fid* for the enroute time of the next instance of a specific flight.

The initial value of an **Hete** for a specific flight is set to the **Ete** of the first instance of the flight.

As flights are flushed and their information is available to update the *GTPS* category database's **Hete** predictions, each new flight instance's actual time enroute (**Ete**) is averaged with the existing value of **Hete**, as follows:

Hete [pending] = 0.95***Hete** [previous] + 0.05***Ete** [of new instance]

If **Ete** exceeds **MaxEte**, currently set to **999** minutes, then the prediction, **Hete**, is not adjusted by this instance's **Ete**.

Error Conditions and Handling

Errors are reported to the user's screen along with a diagnostic message. If the error is terminal, the program exits, allowing the user to investigate and correct the problem.

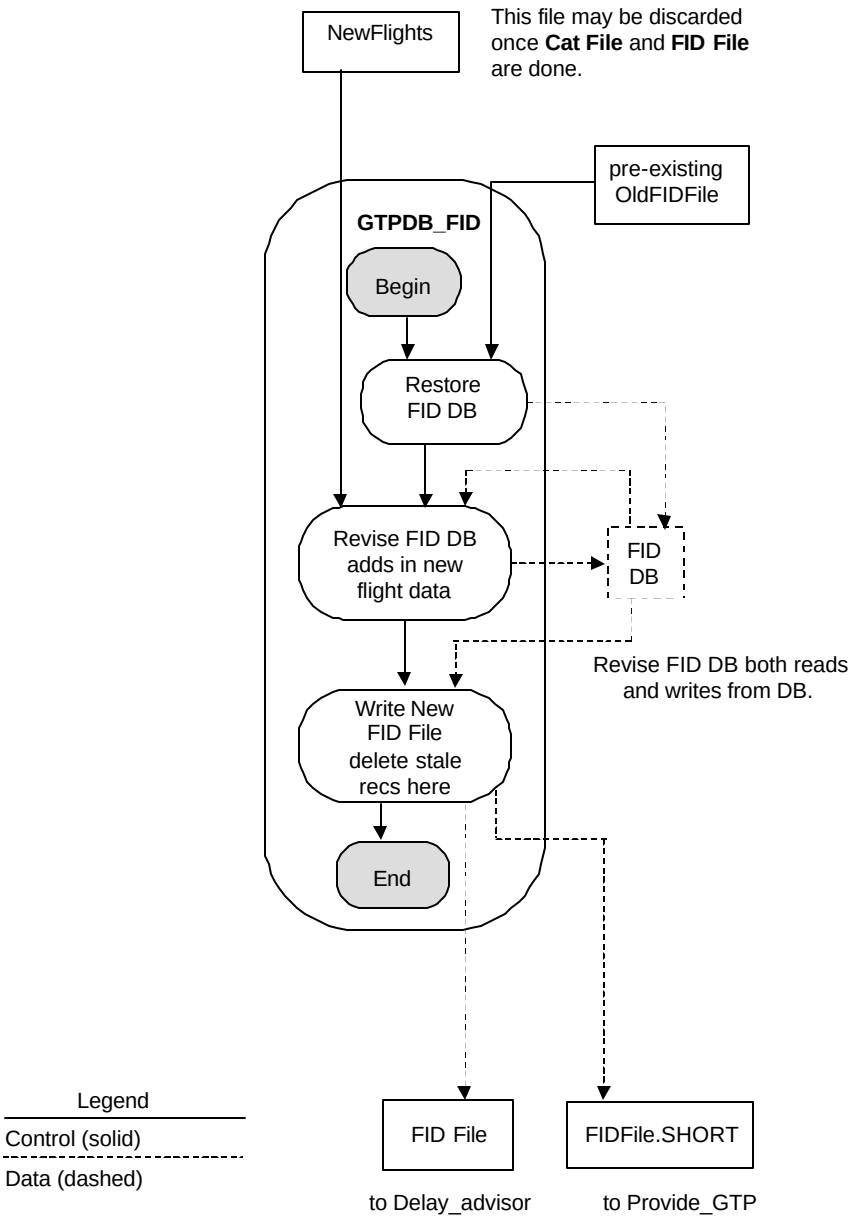

Figure 28.7. Data Flow of the gtpdb_fid Process

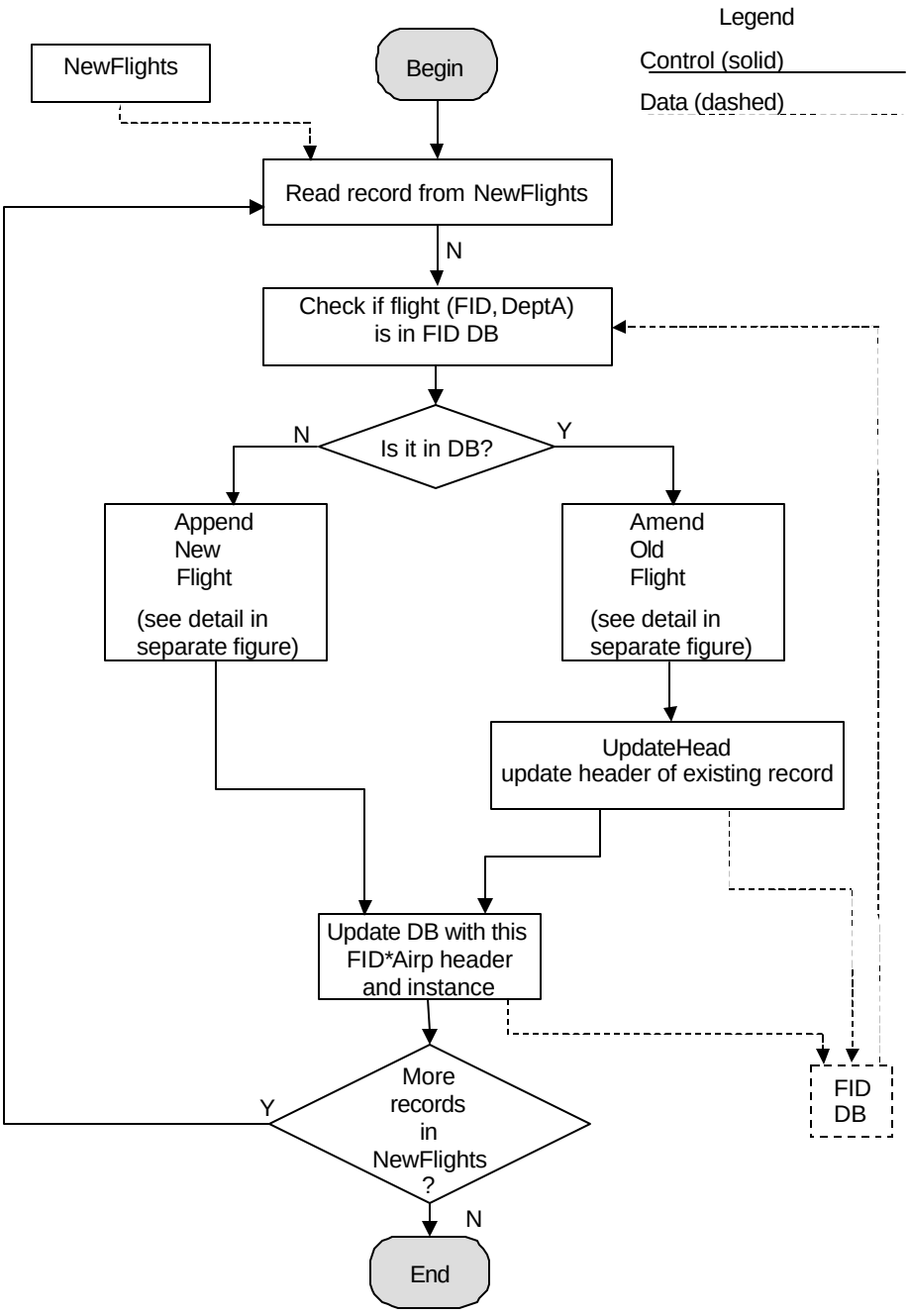

Figure 28.8. Logic for Revise_FID_DB, Part One

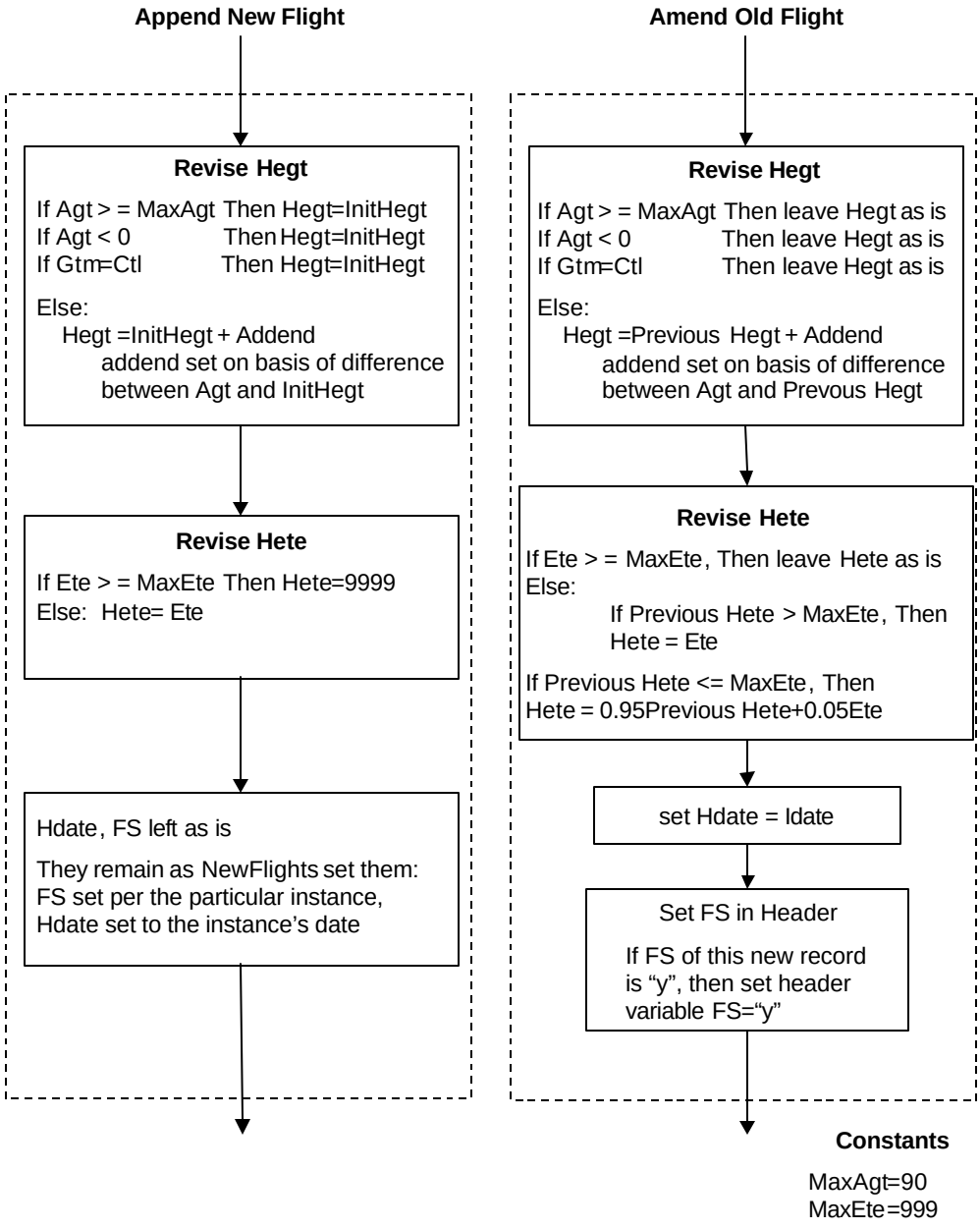

Figure 28.9. Logic for Revise_FID_DB, Part Two

28.2.3 The gtpdb_cat Process

Purpose

The *gtpdb_cat* process creates or revises a database of ground times, organized by generic flight departure categories. The process writes the database to a file named **catfile**. For each category, the process makes a prediction of the ground time expected of the next instance of that category of flight to depart. Information about individual flights or flight instances is not retained, only a revised categorical prediction is retained. The categories partition the set of all flights by airport, departure time of day, departure day of week, and flight duration. Only airports listed in the **cat_def_file** generated by the *build_def_file* process have categories in this database. Controlled flight instances and instances with negative or outlying ground times do not contribute to this database.

Design Issue

The category database is maintained as a simple array. The array is four dimensional, with one index representing the airport (currently 55 are listed), the second index the day of week, the third the time of day category, and the fourth the duration category. The value of the array entry is the predicted ground time associated with the category defined by the concatenation of the array indices.

Execution Level

This process is run in batch mode, at a nominal frequency of once weekly. It is run by entering the following command string:

<executable name (e.g., bpt_cat.exe)> <restoration file name (e.g., old_catfile)>
<revision file name (e.g., newflights)> <path name for directory for output files>

The user is prompted to enter the path name of the working directory at this time. The program assumes that a **restoration** file (a previous incarnation of **catfile**), if any, will be found in the user-specified working directory. The program will place its output, the **catfile**, in that same directory.

Input

The process uses the following two input files:

(1) The previous output file from this process is an old version of the **catfile**. This input is used to restore the category database as it was at the end of the last execution of the process.

(2) A **newflights** file, which is used to revise the restored database in accordance with data collected since the last execution of the process.

Output

The process output is **catfile**, an updated binary file of flight data that is organized into categories.

Processing

The *gtpdb_cat* process maintains and updates a weighted running average of the ground time experienced by each category of flight departures. It uses this average as a prediction of the ground time of the next flight instance of the same category. The *gtpdb_cat* process does the following:

- Restores the category database that had been previously generated
- Revises the category database with new flight data recently collected
- Writes the revised database to a new **catfile** **Catfile** does not become stale and is not discarded. As new flights arrive to increase the population that has contributed to a category, the older flights have a lesser influence on the ground time prediction.

Figure 28-10 presents an overview of the *gtpdb_cat* process, after the text description of the update algorithm. Figure 28-11 shows more detail about the *revise_cat_db* aspects of the process.

Prediction of Ground Time Based on Category

Categorical or generic predictions of ground time, denoted as **Gd**, are made for each of various categories of flights. These categories are defined in terms of ranges of time of day, duration of flight, day of week, and specific airport. Categorical predictions are made for each of 55 key airports. Categorical predictions are not made for other airports.

When *gtpdb_cat* receives a flight from the data archive that is the first flight belonging to one of the database's categories, the process arbitrarily assigns an initial value, **InitCatPred**, for the ground time prediction associated with the category. The current value of **InitCatPred** is 15 minutes. **InitCatPred** becomes the initial "previous" value of **Gd**. That initial value is then

immediately updated by use of the flight's own ground time in accordance with the update equation.

As a flight is flushed, its actual ground time (**Agt**) is averaged with the existing value of **Gd** to obtain an updated predicted value, shown as follows:

Gd [pending] = 0.95***Gd** [previous] + 0.05***Agt** [of new instance]

The following classes of flights are not used to update the ground time running average maintained for each category:

- Controlled flights
- Flights with a negative actual ground time, **Agt**
- Outliers, flights with **Agt** exceeding **MaxAgt**

  (current value of **MaxAgt** = 90 minutes)
- General Aviation flights

Error Conditions and Handling

Errors are reported to the user's screen along with a diagnostic message. If the error is terminal, the program exits, allowing the user to investigate and correct the problem.

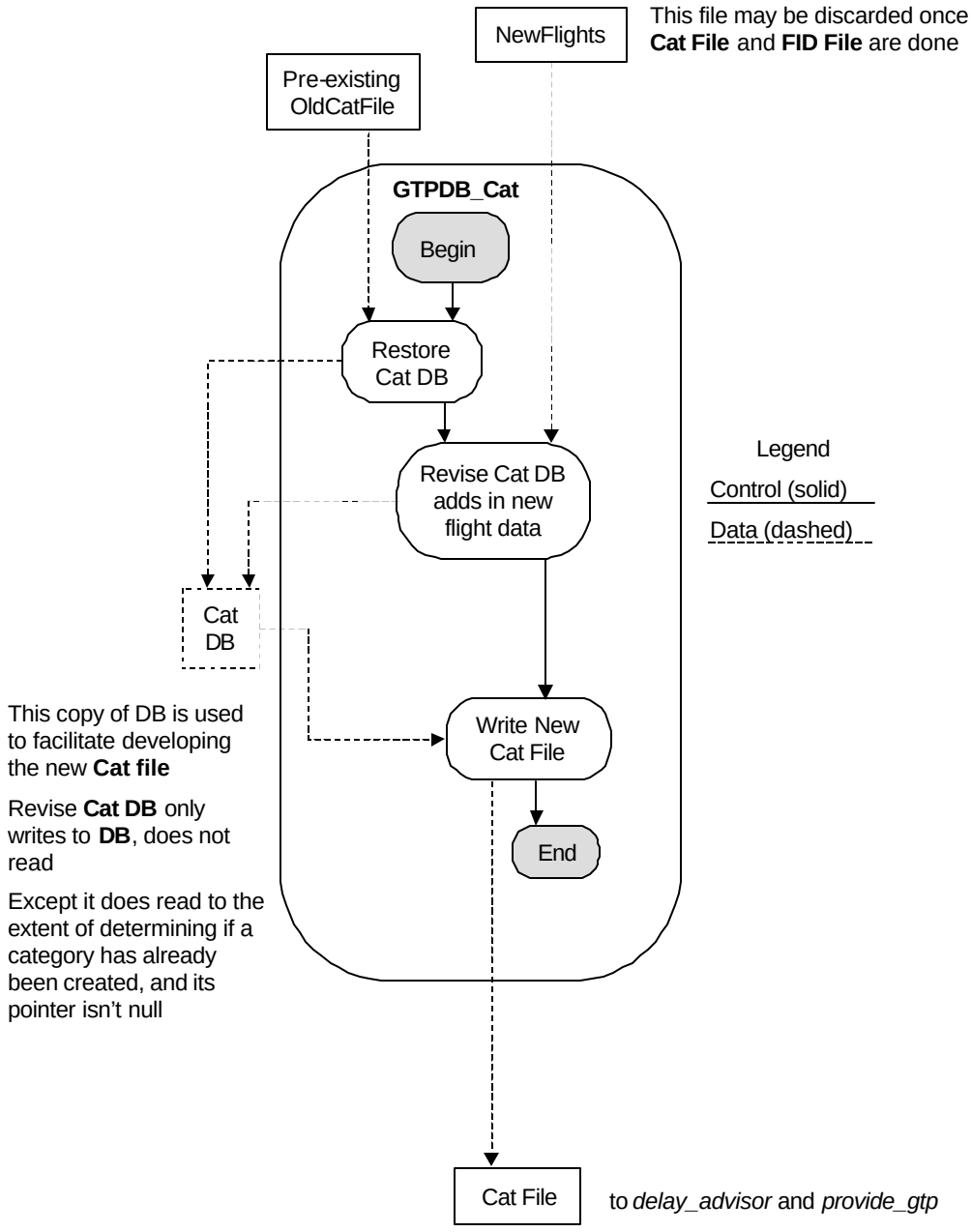

Figure 28.10. Logic for the gtpdb_cat Process

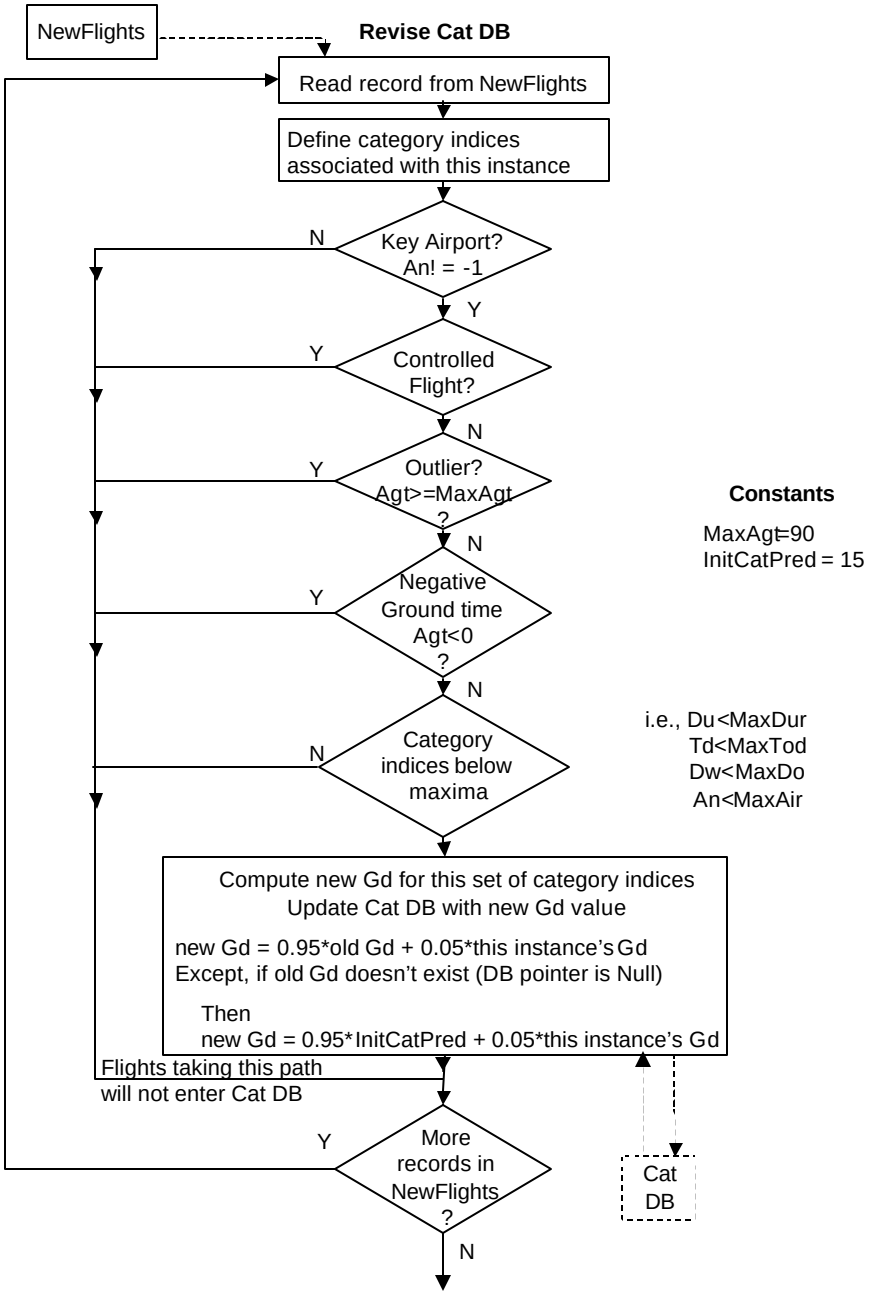

Figure 28.11. Logic for Revise Cat DB

28.3 The provide_gtp Subfunction

The *provide_gtp* subfunction is embedded in the *FDBP*. The *provide_gtp* subfunction provides ground time predictions for flights as the *FDB* needs them. It then takes those predictions from a category-based or a FID-based database in accordance with the algorithm described in Figure 28-2. his subfunction is implemented by the module *provide_gtp* which is a part of the *FDB* process.

28.3.1 The provide_gtp module

Purpose

This module provides the *FDBP* with ground time predictions (and an indication of the method by which the prediction was made) upon request.

Design Issue

Provide_gtp was designed with the intention of minimizing its impact on the *FDBP*. For determining a prediction, an algorithm looks up a prepared prediction in the category and FID prediction databases; it does not itself compute a prediction. Only those parts of the algorithm that are needed are used for the particular prediction to be made. The simplest and most likely cases are attempted first and, if successful, subsequent parts of the algorithm are not used.

Execution Level

This module is used by the *FDBP*, which runs continuously. The *FDBP* invokes *provide_gtp* whenever the *FDBP* processes an FS or FZ message.

Input

The *provide_gtp* module requires a **catfile**, **fidfile.short**, and the **cat_def_file**.

At initialization of the *FDBP*, p*rovide_gtp* reads the information from the **cat_def_file** into its memory. During operations, the **catfile** and **fidfile.short** provide data inputs from which *provide_gtp* will build prediction databases. In those cases for which flight history data is unavailable, *provide_gtp* uses the **cat_def_file** information in deciding to which category a flight pertains.

Output

Output from *provide_gtp* are ground time predictions and the method of prediction. These are given to the *FDBP* for assignment to flight records.

Processing

At *FDBP* initialization, the *provide_gtp* module reads into memory the **cat_def_file**, and uses the *bldcatdb* and *bldfiddb* routines to restore to memory current versions of the category and FID databases.

In operation, *provide_gtp* receives requests for ground time predictions. It responds to them by carrying out the algorithm described in Figure 28-13. Figures 28-14, 28-15 and 28-16 show the logic by which *provide_gtp* obtains a prediction.

Criteria for Switching Between FID-Based and Category-Based Predictions

The FID-based prediction is used in preference to the category-based whenever possible. The exception in which the category-based prediction is used is that the flight (i.e., the concatenation of FID and departure airport) is not listed in the flight database.

If a flight is listed in the FID database, but the flight has controlled status, then a default of zero minutes is used for the prediction.

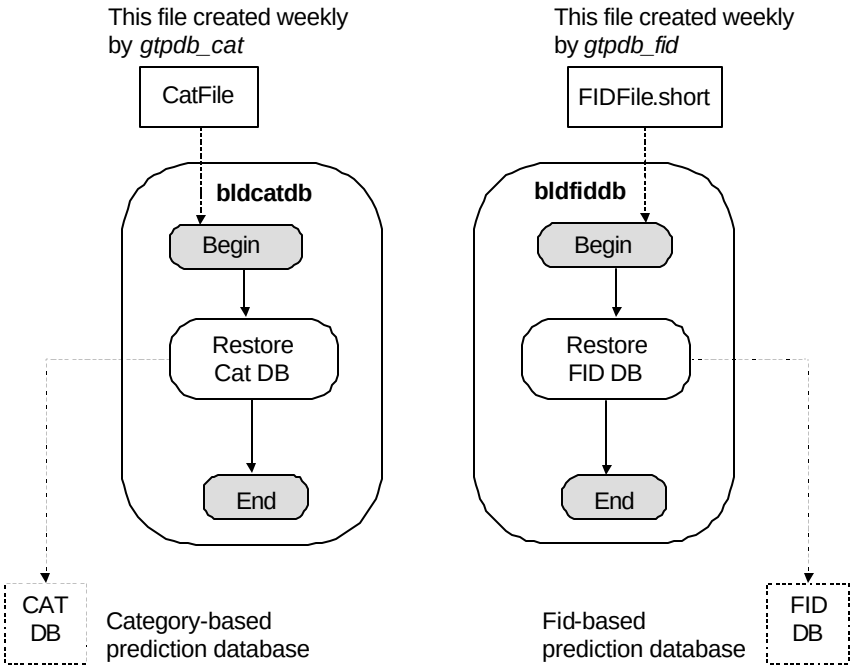

Figure 28-12. Building the Prediction Databases for Provide_GTP

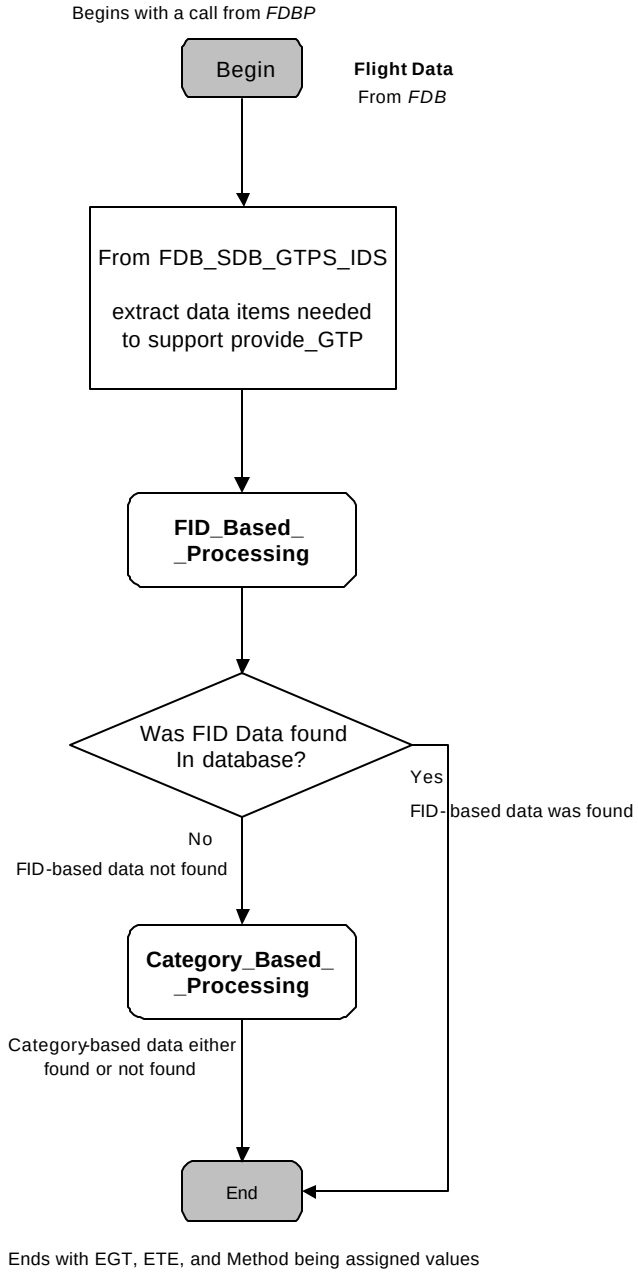

Figure 28-13. Logic for Provide_GTP

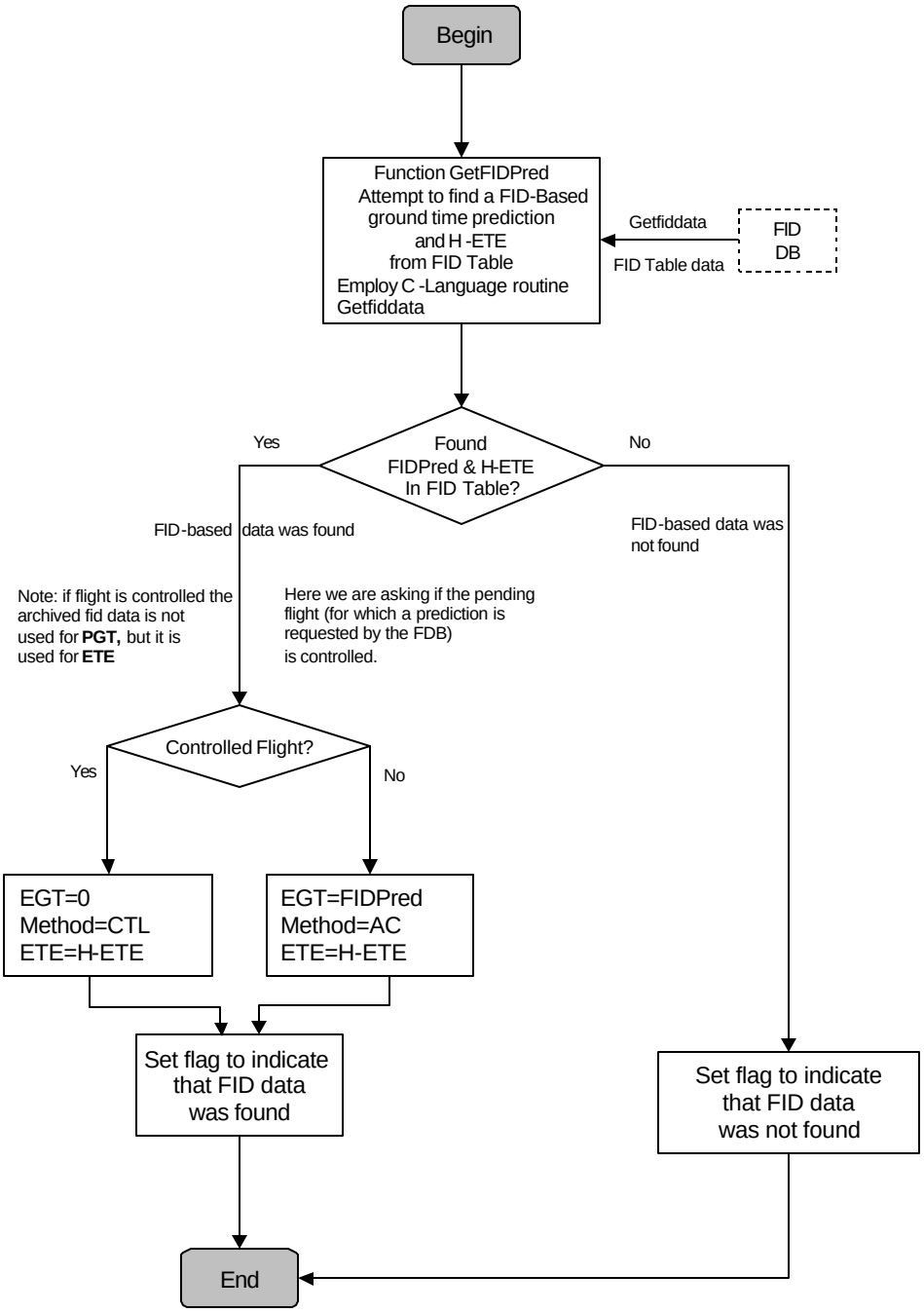

Figure 28-14. Logic for FID_Based_Processing

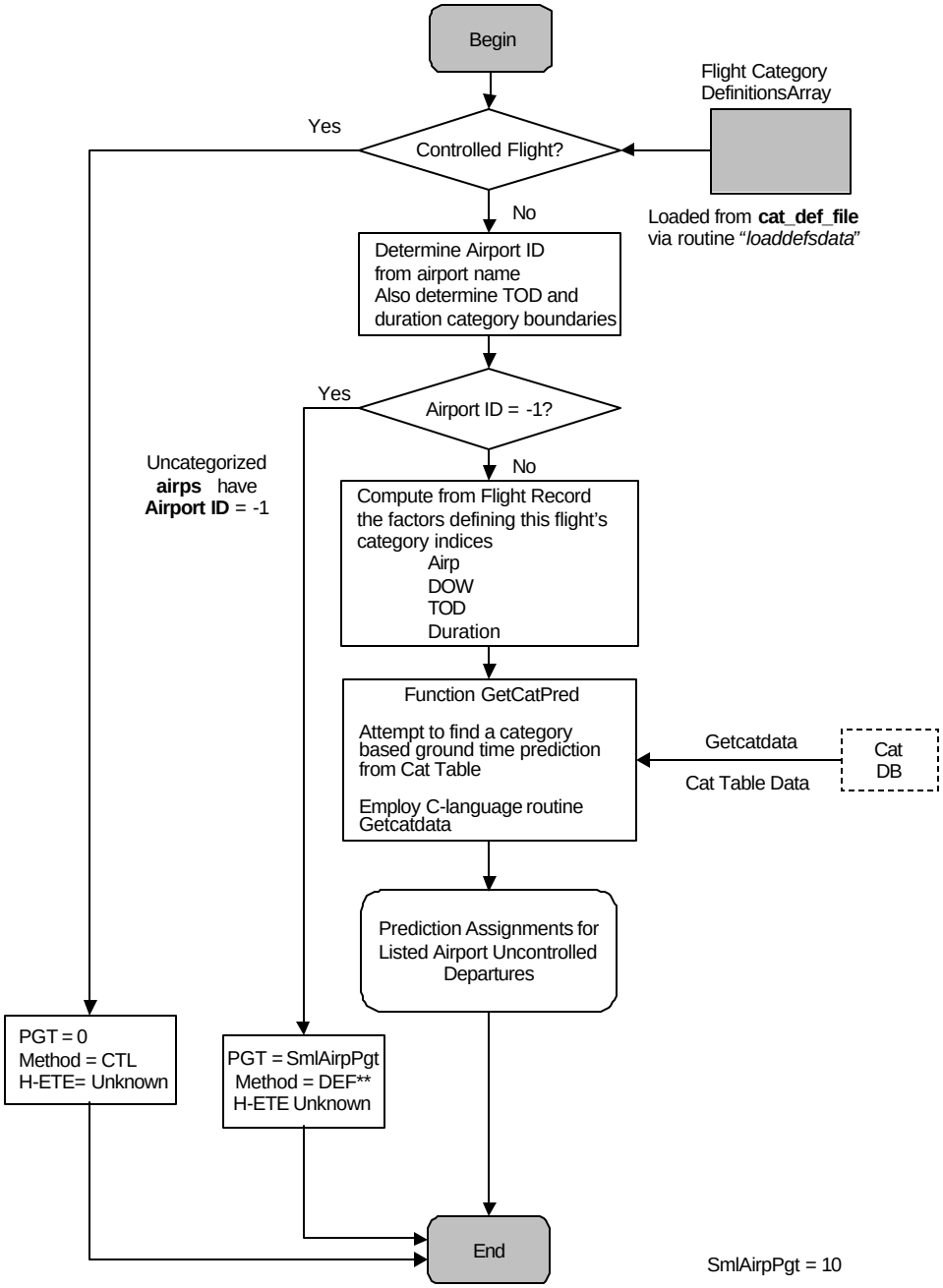

Figure 28-15. Logic for Category_Based Processing

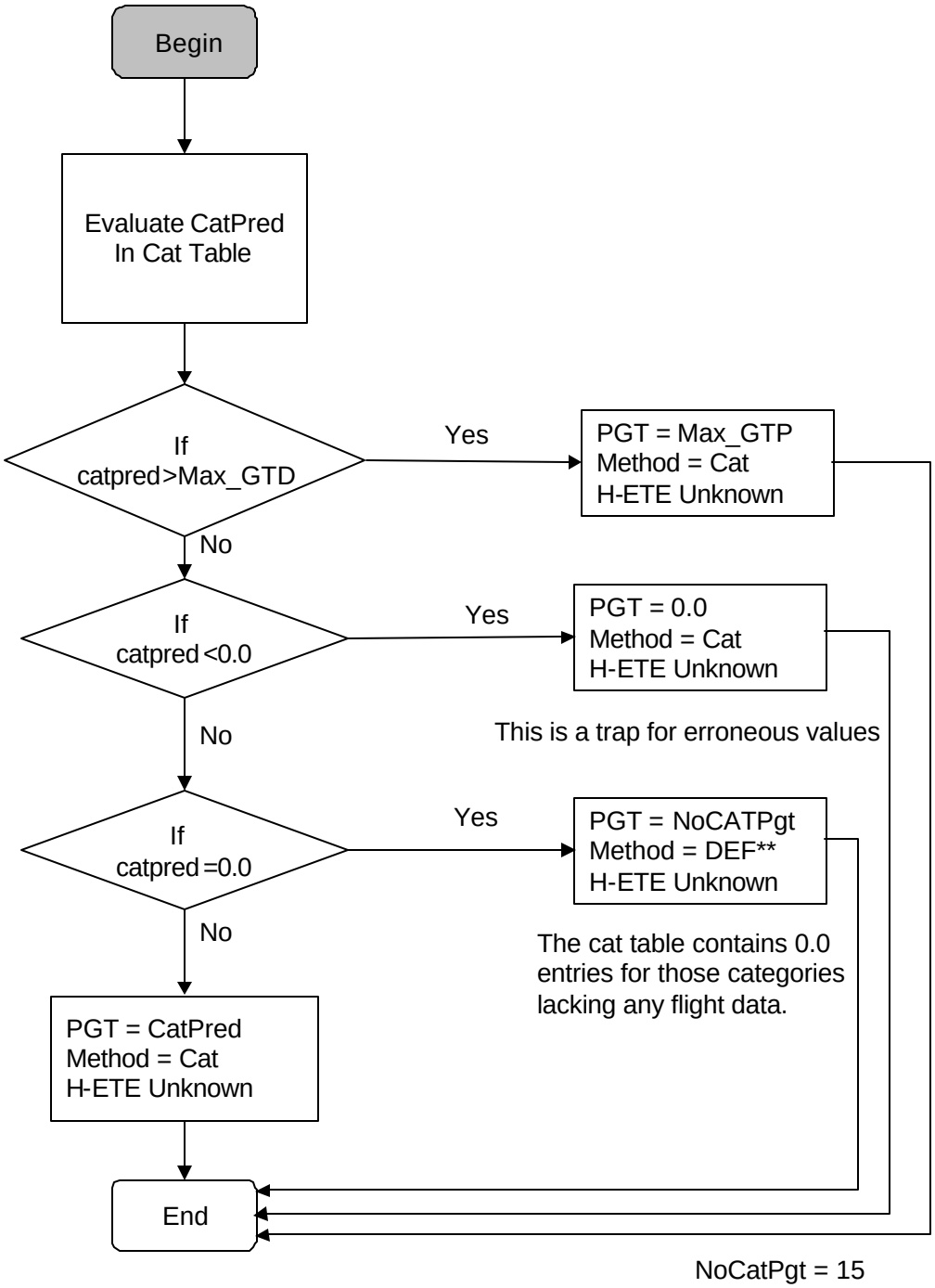

Figure 28-16. Prediction Assignments for Listed Airport Uncontrolled Departures

28.4 The build_def_file Process

Purpose

The *build_def_file* ancillary process is used as a convenience for preparing the **cat_def_file** (see Figure 28-17). This file lists the key airports for which category predictions will be made and for which reports describing ground times by category will be available. The *build_def_file* process does the following:

- Assigns to each airport an arbitrary identifier number
- Assigns to each airport the number of time of day and duration categories
- Defines the boundaries of those categories
- Writes the data to the **cat_def_file**

This data is later used by the processes comprising *build_prediction_tables* and by the *delay_advisor* subfunction.

Design Issue

For convenience in development, debugging, testing, and maintenance, it was decided that the **cat_def_file** would be an ASCII file.

Execution Level

This process is run in batch mode and is only run when ETMS program management determines that some change is needed in the **cat_def_ file**, which is the process output.

Input

None. The information used by this process is hard-coded into it. Input is provided under the auspices of the ETMS program management.

Output

Build_def_file produces the file named **cat_def_file**. This file contains data needed by other *GTPS* processes to identify airports and to help categorize by duration and time of day flights departing those airports.

Processing

The process directly lists the key airport names. It uses simple assignment statements to assign each airport an identifying index and assign duration and time of day category boundaries unique to the airport.

Error Conditions and Handling

Errors are reported to the user's screen along with a diagnostic message. If the error is terminal, the program exits, allowing the user to investigate and correct the problem.

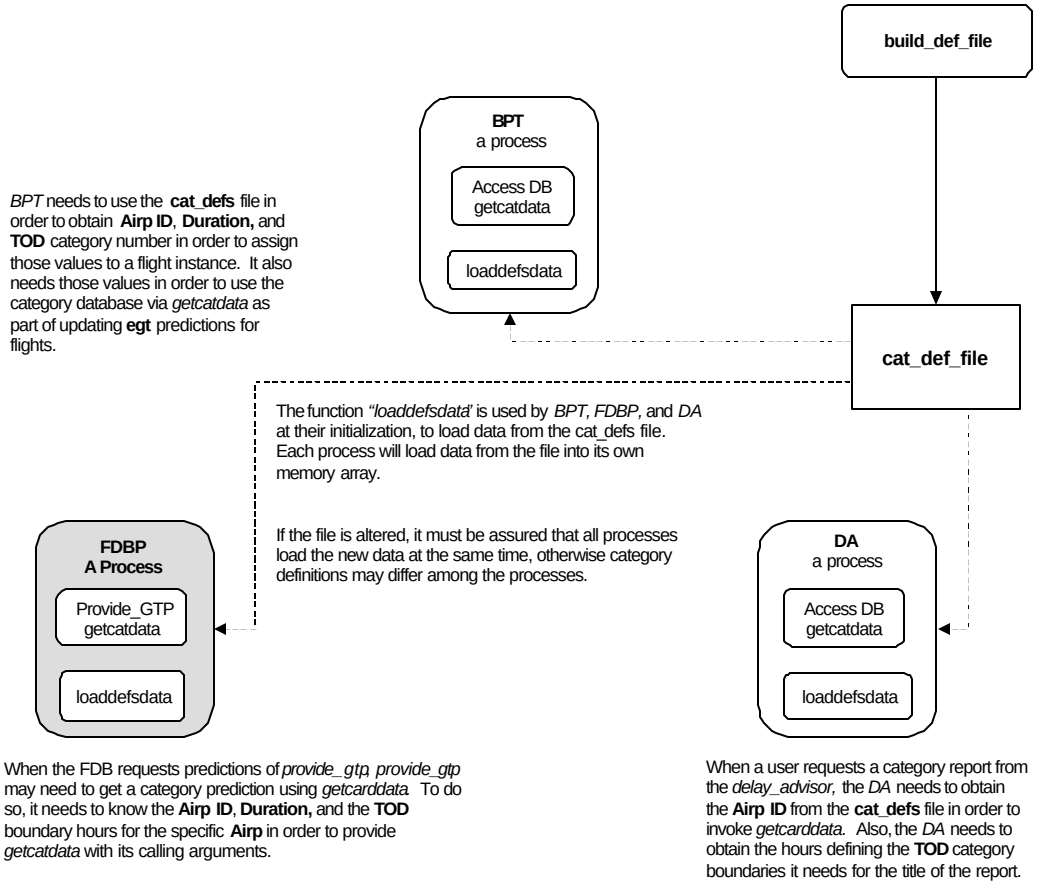

Figure 28-17. Usage of the cat_def_file

28.5 GTPS Test Routines

Three test routines were prepared to facilitate debugging and testing. These routines allow a user to print and read the **newflights** file, the **fidfile** (or **fidfile.short**), and the **catfile**, which are unprintable binary files in their native form. See Figure 28-18.

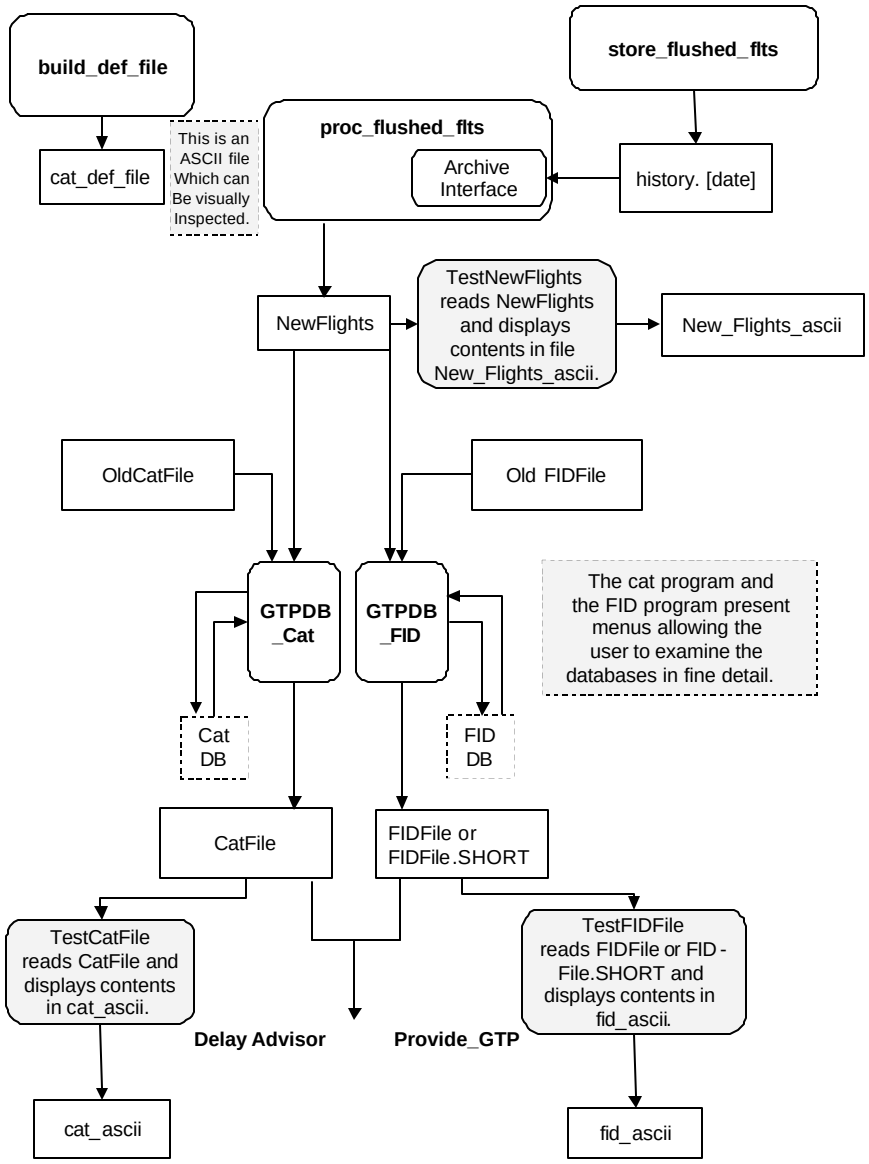

Figure 28-18. Built-in Test Features

A design issue for each of the test routines is that the output ASCII file is to be as equivalent as possible to the input binary file, except that only the output is printable and user-readable. There is no intent to filter any of the information, or apply any decision criteria to it.

28.5.1 The testnewflights Test Routine

Purpose

The *testnewflights* test routine takes the **newflights** file as an input and generates an ASCII equivalent called **NewFlights_ascii**

Execution Level

This routine is executed in batch mode as desired to support debugging and test. The routine is run by entering the following:

testnewflights <name of file of new flights>

Input

The input is any file of the **newflights** file data structure as described by the **newflights** data structure.

Output

The ouput is an ASCII file named **New_Flights_ascii**, of the same apparent data structure as **newflights**.

Processing

The input data structure is decoded and written to an ascii file.

Error Conditions and Handling

Errors are reported to the user's screen along with a diagnostic message. If the error is terminal, the program exits, allowing the user to investigate and correct the problem.

28.5.2 The testcatfile Test Routine

Purpose

The *testcatfile* routine takes the **catfile** file as an input and generates an ASCII equivalent called **catfile_ascii**.

Execution Level

This routine is executed in batch mode, as desired, to support debugging and test. It is run by entering the following:

testcatfile <name of file catfile>

Input

The input is any file of the **catfile** file data structure.

Output

The ouput is an ASCII file named **catfile_ascii**, of the same apparent data structure as **catfile**.

Processing

The input data structure is decoded and written to an ASCII file named **catfile_ascii**.

Error Conditions and Handling

Errors are reported to the user's screen along with a diagnostic message. If the error is terminal, the program exits, allowing the user to investigate and correct the problem.

28.5.3 The testfidfile Test Routine

Purpose

This routine takes the **fidfile** or the **fildfile.short** file as an input and generates an ASCII equivalent called **fidfile_ascii**.

Execution Level

This routine is executed in batch mode as desired to support debugging and testing. It is run by entering

testfidfile <name of file fidfile>

Input

The input is any file of the **fidfile** file data structure, including possibly **fidfile.short**.

Output

The ouput is an ASCII file named **fidfile_ascii**, of the same apparent data structure as **fidfile**.

Processing

The input data structure is decoded and written to an ASCII file **fidfile_ascii**.

Error Conditions and Handling

Errors are reported to the user's screen along with a diagnostic message. If the error is terminal, the program exits, allowing the user to investigate and correct the problem.

28.6 GTPS Data Structures

This section describes the data structures used by the various processes comprising the *GTPS* subsystem. Some of these structures are used by one and only one process. Some are shared between the processes, and some are shared with other ETMS functions which interact with the *GTPS* via these data structures.

The *GTPS* employs five key data structures:

(1) **history_file**

(2) **newflights**

(3) **fidfile**

(4) **catfile**

(5) **cat_def_file**

The **history_file** and the **cat_def_file** are ASCII files and the others are binary files. They are defined in the Tables 28-2 through 28-6 at the end of this section.

In addition to the five key data structures, there are three ancillary structures generated by the test routines described in Section 28.5. These ancillary structures are **newflights_ascii**,

fidfile_ascii, and **catfile_ascii**, also described in the tables at the end of this section.

28.6.3 The history_file Data Structure

This structure is an ASCII file.

The **history_file** data structure is used to store records of flushed flight information (see Table 28-2). This information is stored in the **history_file** as it is flushed. It is later used by *Build Prediction Tables* as the material from which it will build prediction tables for subsequent flights.

Table 28-2. History_file Data Structure

Store Flushed Flights Output Content			
Purpose: This is the data structure of the file history.[date], which is generated by the store_flushed_flts.c and used by proc_flushed_flts.c			
Data Item	**Definition**	**Unit/Format**	**Var. Type**
id_of_flight	flight identification	1 or 3 letters followed by numbers	string7
departure_date	julian date for flight's departure	number of days since January 1, 1980	unsigned short
user_category	category of user	an enumerated type: commercial, military, etc.	usercat_t
ac_cat_class	specific type of aircraft	an enumerated type: civilian jet, single_piston_prop, etc.	ac_cat_t
ac_weight_class	aircraft weight class	an enurmerated type: small, large, etc.	ac_weight_t
ground_time	predicted number of minutes for flight to taxi	minutes	short
ground_time_method	mode used in ground time determination	a=aircraft; c=category; t=controlled; d=default	char
departure_ap	airport of flight's origin	combination of 3 or 4 letters and numbers	string4
sched_dep_time	flight's scheduled departure time (from	minutes from midnight	short
proposed_dep_time	flight's proposed departure time (from	minutes from midnight	short
control_dep_time	flight's controlled departure time (from EDCT)	minutes from midnight	short
actual_dep_time	flight's actual departure time (from DZ)	minutes from midnight	short
arrival_ap	destination airport of flight	combination of 3 or 4 letters and numbers	string 4
sched_arr_time	flight's scheduled time of arrival (from FS)	minutes from midnight	short
proposed_arr_time	flight's propsed arrival time (from FZ)	minutes from midnight	short
control_arr_time	flight's controlled arrival time	minutes from midnight	short
curr_arr_time	current prediction for flight's arrival time	minutes from midnight	short
actype	NAS abbreviation for aircraft type	combination of letters and numbers	string4

28.6.4 The newflights Data Structure

This structure is a binary file, and is generated by *proc_flushed_flts.c.*

The **newflights** data structure is used as an intermediate file (see Table 28-3). It holds the information from the history file, except irrelevant or unusable data is eliminated, and certain derived variables are appended to each flight's record. This file serves as an input for the construction of the **fidfile** and **catfile**.

The newflights_ascii Data Structure

This structure is an ASCII file.

The **newflights_ascii** data structure is an ascii representation of the **newflights** data structure. It is intended as a convenience to development, testing, and revision.

28.6.5 The fidfile and fidfile.short Data Structures

These structures are binary files and are generated by *gtpdb_fid.c.*

The **fidfile** data structure is used to hold information about individual flights (see Table 28-4). It is organized as an array of linked lists; each linked list consists of the 14 most recent instances of a particular flight. It holds information directly taken from the **newflights** file as well as predicted values of ground times and enroute times which are computed algorithmically. This file serves as a source of ground time and enroute time predictions to traffic specialists via the *delay advisor*.

Fidfile.short has the same structure as **fidfile**. **Fidfile.short** differs from **fidfile** in the following two respects:

(1) It contains only the most recent instance per flight, not the last 14 instances.

(2) It excludes flights for which the **fidfile** has fewer than minimum (nominally 4) fid instances. This file serves as a source of ground time predictions to the FDB via the *provide_gtp* process.

The fidfile_ascii Data Structure

This structure is an ASCII file.

The **fidfile_ascii** data structure is an ASCII representation of the **fidfile** data structure. It is intended as a convenience to development, testing, and revision.

Table 28-3. Newflights Data Structure

Data Structure Name: output_rec_t		
Purpose: This is the data structure of the **newflights** file, which is generated by proc_flushed_flts.c. **newflights** is used by gtpdb_fid.c and gtpdb_cat.c; but each of these two programs only uses a subset of the data items of **output_rec_t**. A number of items are assigned values for this data structure simply to hold a place, facilitating subsequent data processing.		
FID Tables Header		
Fid	Flight Identifier, read directly from history file	Fid_t
DepA	Departure airport, read directly from history file	DepA_t
DestA	Destination airport, read directly from history file	DepA_t
FidCount	Assigned placeholder value = 1	un.short
Hegt	Assigned placeholder value = Egt	un.short
Hete	Assigned placeholder value = Ete	un.short
Hdate	Assigned placeholder value = Idate	un.short
Fs	Flight Schedule Flag, set to y if there exists a sched dept; otherwise to n	char
FID Tables Instance		
Egt	Predicted ground time, read directly from history file	un.short
Agdt	MinFromMid of (actl. dep. time)	un.short
Pgdt	MinFromMid (ctld), or MinFromMid (prop), or 9999	un.short
Gtm	Ground time method, read directly from history file	un.short
Idate	JulianDate of this instance read from history file	un.short
Ete	Estimated time enroute, from history file values, [actu arr – actu dept]	un.short
Other Item		
Agt	Agdt-Pgdt; [except if Pgdt =9999, Agt=9999; if Agt<MinAgt, Agt=9999]	short
Cat Table Items		
An	Departure airport ID from cat_defs_file	short
Du	Duration category index from cat_def_file, based on Ete	short
Td	Time of day category index from cat_def_file, based on Pgdt	short
Dw	Day of week category index from cat_def_file, based on Idate	short
Gd	Assigned placeholder value = Agt	float

Note: Gtm values 0,1,2,3,4 correspond to ac, cat, ctl, default, spare.
**Defaults of 9999 are used for unavailable fields

Table 28-4. Fidfile Data Structure

Data Structure Name: output_rec1_t		
Purpose: This is the data structure of the file **fidfile** and **fildfile.short**, generated by *gtpdb_fid.c*.		
Fid Tables Header		
Fid	Flight Identifier, read directly from history file	Fid_t
DepA	Departure airport, read directly from history file	DepA_t
DestA	Destination airport, read directly from history file	DepA_t
FidCount	Count of instances of this FID—DepA	un.short
Hegt	Current prediction of ground time for this flight	un.short
Hete	Current prediction of time enroute for this flight	un.short
Hdate	Most recent julian date of occurrence of this flight	un.short
Fs	Flight Schedule Flag	char
Fid Tables Instance		
Egt	This instance's predicted ground time, from history file	un.short
Agdt	MinFromMid of (actl.dep.time)	un.short
Pgdt	MinFromMid (ctld), or MinFromMid (prop), or 9999	un.short
Gtm	Ground time method, from history file	un.short
Idate	JulianDate of this instance read from history file	un.short

28.6.6 The catfile Data Structure

This structure is a binary file and is generated by *gtpdb_cat.c*.

The **catfile** data structure is used to hold information about categories of flights (See Table 28-5). It is organized as a four-dimensional array with dimensions of airport, day of week, duration of flight, and departure time of day. It holds information directly taken from the **newflights** file as well as predicted values of ground times which are algorithmically computed. This file serves as a source of ground time and enroute time predictions to the FDB via *provide_gtp*, and to traffic specialists via the *delay advisor*.

The catfile_ascii Data Structure

This structure is an ASCII file.

The **catfile_ascii** data structure is an ASCII representation of the **catfile** data structure. It is intended as a convenience to development, testing, and revision.

Table 28-5. Catfile Data Structure

Data Structure Name: Air[][][][]		
Purpose: This is the data structure of the file **CatFile**, which is generated by *gtpdb_cat.c*. Air is a four dimensional array: Air [Airport ID] [DOW index] [TOD index] [Duration index] = Gd		
An	Departure airport ID from **cat_defs_file**	short
Du	Duration category index from **cat_def_file**, based on **Ete**	short
Td	Time of day category index from **cat_def_file**, based on **Pgdt**	short
Dw	Day of week category index from **cat_def_file**, based on **Idate**	short
Gd	Predicted ground time of this category's flights, from running average	float

**Defaults of 9999 are used for unavailable fields

28.6.7 The cat_def_file Data Structure

This structure is an ASCII file.

The **cat_def_file** data structure identifies the key airports for which category predictions will be made and for which reports describing ground times by category will be available (see Table 28-6). The file identifies the key airports simply by naming each of them. The file assigns to each airport an arbitrary identifier number. It assigns to each airport the number of time of day and duration categories and it defines the boundary conditions of those categories. This data is later used by the processes comprising build prediction tables, it is also used by the delay advisor process.

Table 28-6. cat_def_file Data Structure

cat_def_file content			
Purpose: This is the data structure of the ancillary file **cat_def_file**, which is generated by build_def_file, and used by build_prediction_tables, delay_advisor, and provide_gtp.			
Data Item	**Definition**	**Unit/Format**	**Var. Type**
airport_name	airport of flight's origin	combination of 3 or 4 letters and numbers	string6
airport_id	airport identity number		short
utc_offset	offset of local time at departure airport from UTC	hours * minutes	short
num_tod_int	number of times of day categories in use at this airport	one digit	short
tod_before_midnight	number of time of day categories at this airport prior to UTC midnight	one digit	short
The following two itesm are replicated five times. If some of them are unused, they are given values of "-1"			
tod_start	start time of tod category for this airport	hours * minutes	int
tod_end	end time of tod category for this airport	hours * minutes	int
The following two items are replicated five times. If some of them are unused, they are given values of "-1"			
mam_start	minutes from midnight format	hours * minutes	int
mam_end	minutes from midnight format	hours * minutes	int
num_dur_cats	number of duration categories in use at this airport	one digit	short
The following two items are replicated six times. If some of them are unused, they are given values of "-1"			
dur_cat_min	minimum time duration of this category	one,two,or three digits	int
dur_cat_max	maximum time duration of this category	one,two,or three digits	int